SDCC

Secure PassWorld

Applicazione basata

su **microservizi**

Enrico D'Alessandro

Alessandro De Angelis

Pierpaolo Spaziani

Cos’è e cosa offre

Il progetto realizzato consiste in un’applicazione con architettura a **microservizi**, che fornisce una piattaforma web per la gestione di **password sicure** e dei **tool di supporto alla sicurezza** per le aziende.

Utenti

- Generare **nuove password**
- Gestire le **password salvate**
- Richiedere e gestire richieste di **password condivise** (solo per utenti *Enterprise*)

Aziende

- Utilizzare il tool di **doppia autenticazione.**
- Utilizzare le **password generate in modo condiviso**, per limitare l’acceso ad aree riservate

I microservizi

Login

Permette ad utenti e aziende di autenticarsi ed accedere alla piattaforma

New Password

Permette agli utenti di generare password sicure numeriche, alfanumeriche o composte solo da catterai

Password Manager

Permette agli utenti la gestione delle password personali salvate

Double Authentication

Permette alle aziende di utilizzare il tool di doppia autenticazione

Group Manager

Permette la gestione di gruppi di utenti da parte delle aziende

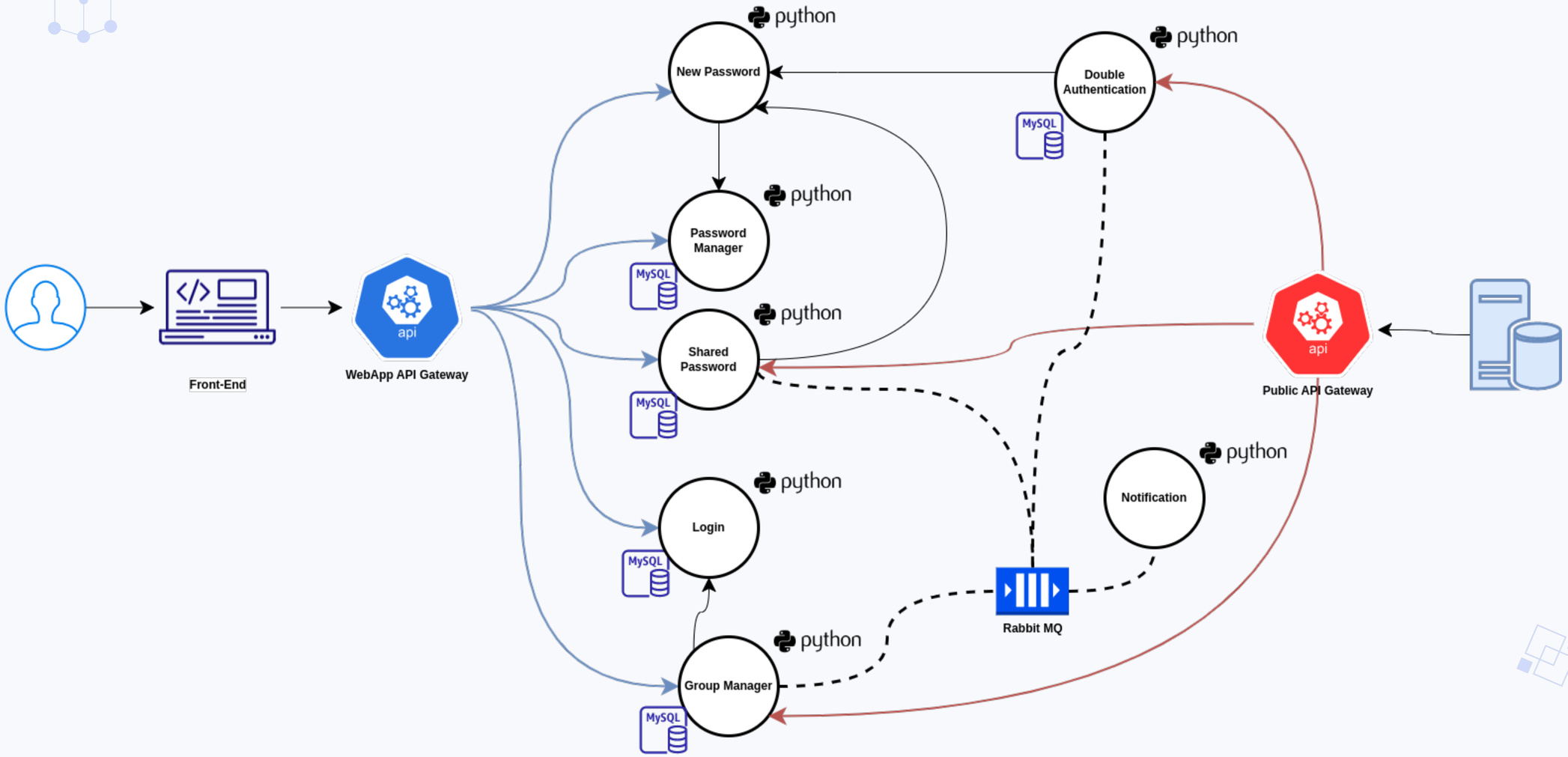
Shared Password

Permette ad utenti *Enterprise* di richiedere ai membri del proprio gruppo una password temporanea

Notification

Permette l'invio di email di notifica agli utenti

Architettura



Secure PassWorld

Comunicazione tra microservizi

Per la comunicazione interna al cluster, sono state adottate le tecniche di **comunicazione sincrona** tramite ***gRPC*** e **asincrona** con ***RabbitMQ***.

gRPC

Tale framework di *Remote-Procedure-Call* è stato utilizzato all'interno dell'applicazione per la comunicazione sincrona tra microservizi

RabbitMQ

RabbitMQ è stato utilizzato come middleware di comunicazione orientato ai messaggi in modo tale da avere disaccoppiamento spaziale e temporale tra i microservizi.

Comunicazione asincrona

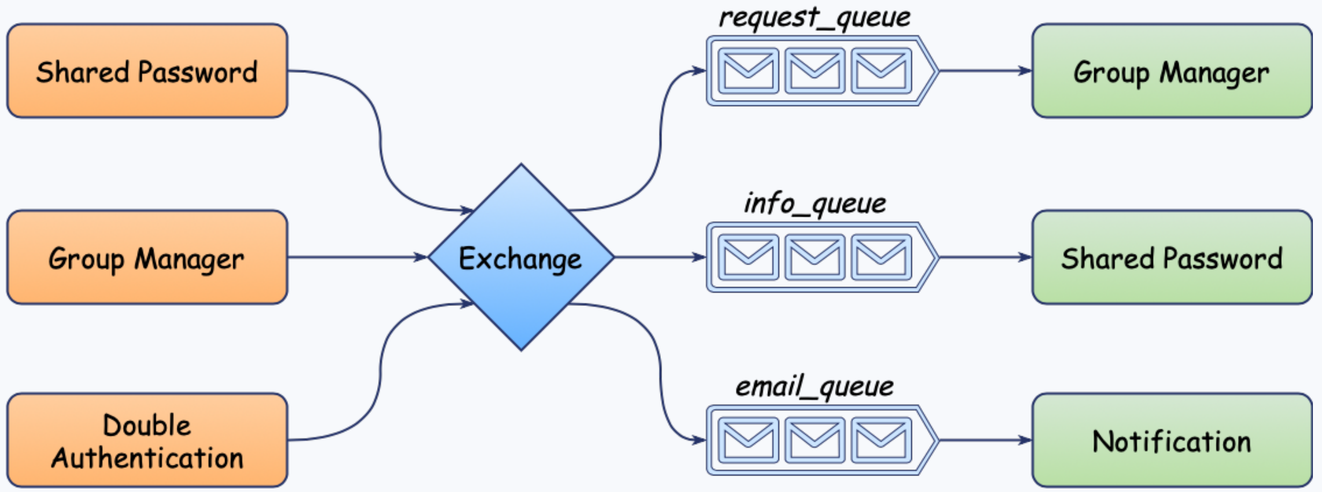
RabbitMQ

Come ***exchange*** è stato utilizzato il tipo ***direct***.

L'algoritmo di routing alla base prevede che un messaggio venga consegnato alla coda corrispondente alla ***routing-key*** specificata dal producer del messaggio.

L'***exchange*** e le ***code*** utilizzate sono state dichiarate come ***durable*** in modo tale da poterle ripristinare in seguito al restart del nodo.

Anche i ***messaggi*** sono stati resi ***persistenti*** in modo tale da non perderli in seguito a crash.

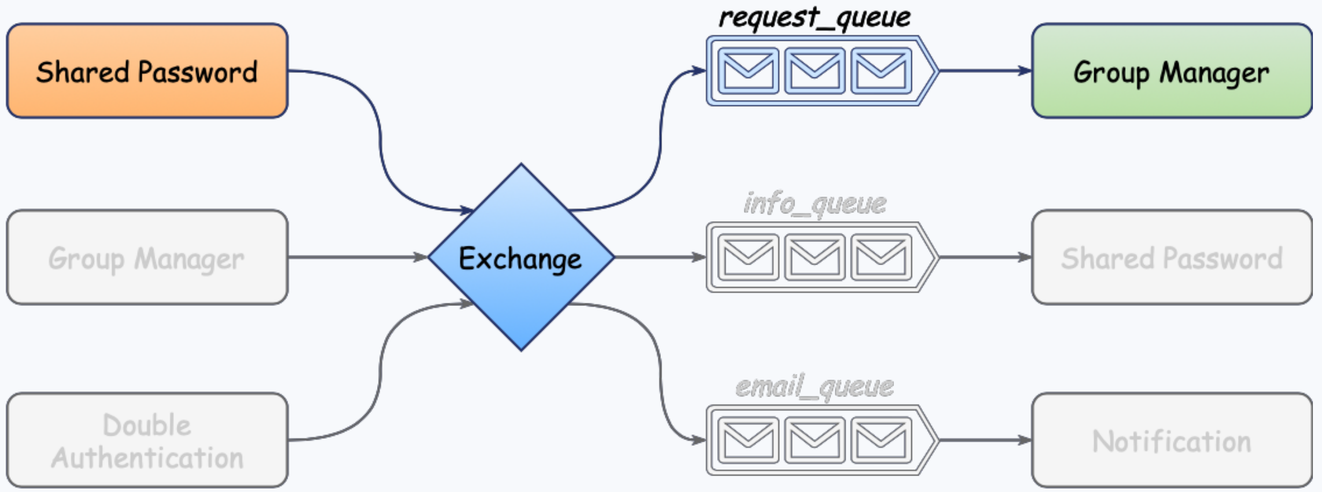


Secure PassWorld

Comunicazione asincrona

RabbitMQ

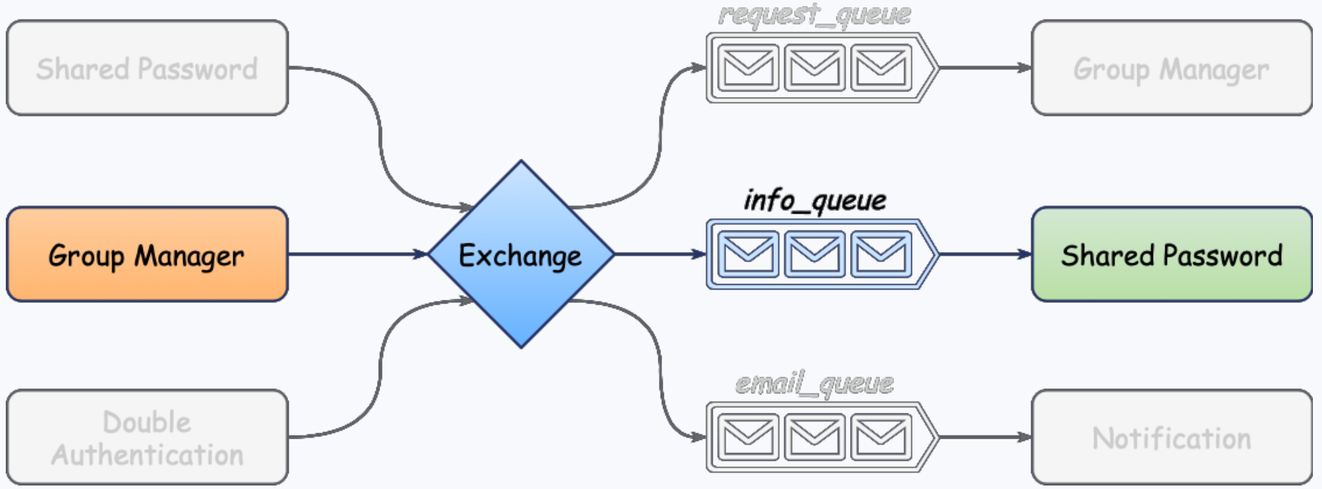
- **Scenario:** Password Condivisa
- **Utilizzo:** Richiesta della lista delle email dei partecipanti ad un specifico gruppo
- **Producer:** Shared Passwords
- **Consumer:** Group Manager



Comunicazione asincrona

RabbitMQ

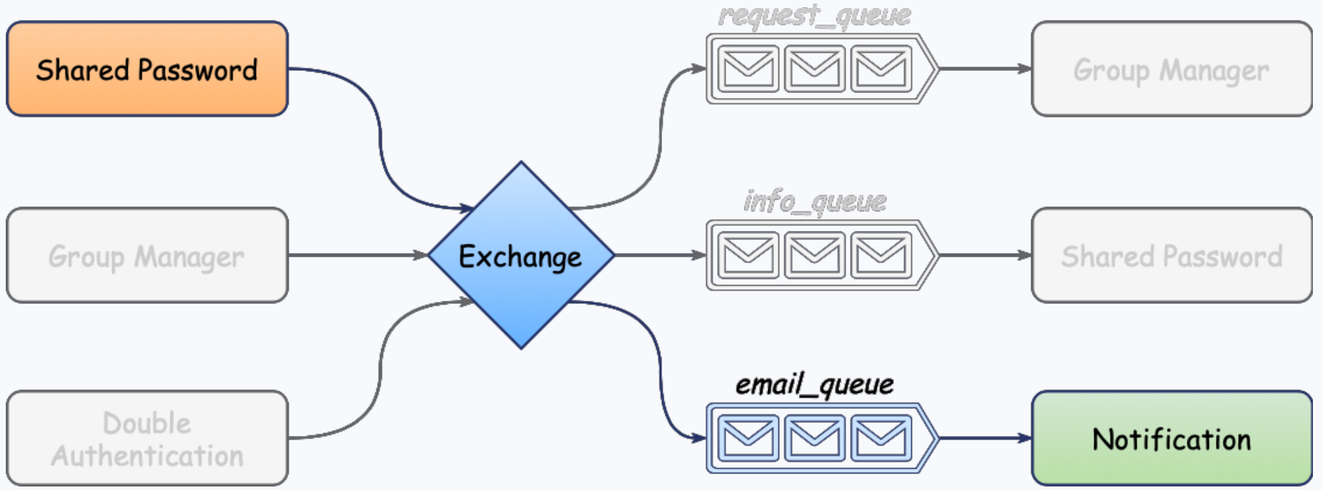
- **Scenario:** Password Condivisa
- **Utilizzo:** Invio della lista di email dei partecipanti al gruppo
- **Producer:** Group Manager
- **Consumer:** Shared Passwords



Comunicazione asincrona

RabbitMQ

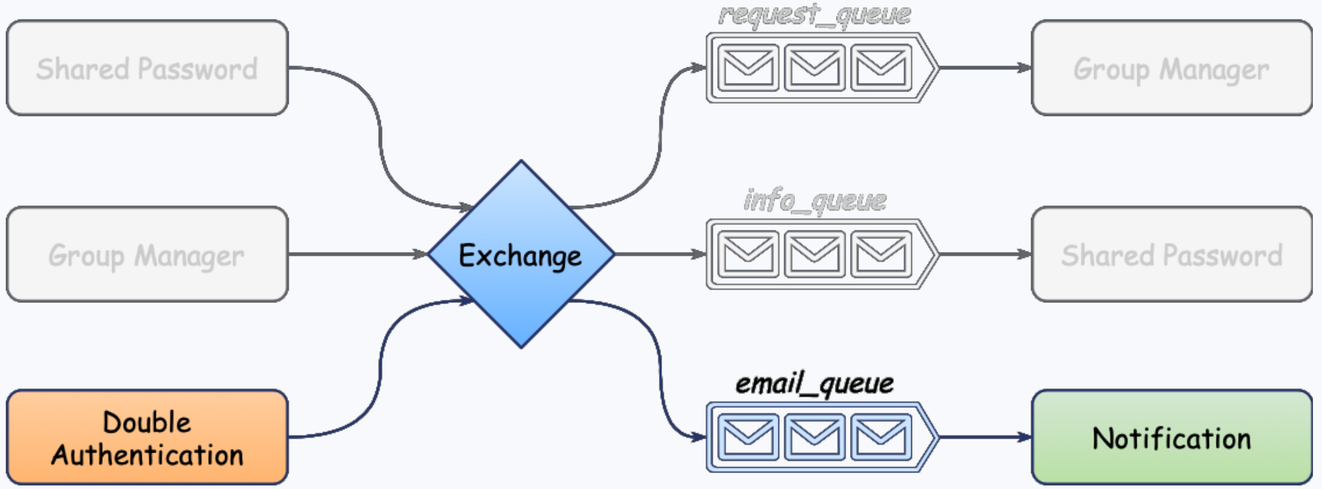
- **Scenario:** Password Condivisa
- **Utilizzo:** Richiesta di invio delle email con i relativi token ai partecipanti del gruppo
- **Producer:** Shared Passwords
- **Consumer:** Notification



Comunicazione asincrona

RabbitMQ

- **Scenario:** Doppia Autenticazione
- **Utilizzo:** Richiesta di invio dell'email contenente il codice di autenticazione all'utente
- **Producer:** Double Authentication
- **Consumer:** Notification



Comunicazione verso l’esterno

L’interazione **verso l’esterno** è stata realizzata tramite l’utilizzo di **due gateway** differenti, ognuno dei quali offre delle **API REST per la comunicazione**.

WebApp

Offre le API per quei microservizi direttamente accessibili dal client attraverso l’interfaccia web, la quale viene gestita da un server web *Flask*.

- **/home**
- **/login**
- **/register**
- **/homepage**
- **/newpassword**
- **/savepassword**
- **/listpassword**
- **/grouplist**
- **/groupcreate**
- **/addEmployee**
- **/notification**
- **/logout**

Comunicazione verso l’esterno

L’interazione **verso l’esterno** è stata realizzata tramite l’utilizzo di **due gateway** differenti, ognuno dei quali offre delle **API REST per la comunicazione**.

PublicAPI

Offre le API per i microservizi utilizzabili da applicazioni terze. Tramite questo gateway, i privati possono sfruttare i servizi offerti a loro: *Doppia Autenticazione* e controllo *Password Condivisa*.

- **/shared_pass**
- **/double_auth**
- **/register_user**

Software utilizzati

Per lo sviluppo ed il testing di Secure Password sono stati utilizzati i seguenti software:

Docker

Consente la **virtualizzazione** a livello di sistema operativo e che permette di creare, testare e distribuire applicazioni con rapidità. L'utilizzo che ne è stato fatto è quello della **creazione di immagini per i container** da instanziare poi all'interno del cluster di *Kubernetes*.

Kubernetes

È una piattaforma per l'**orchestrazione** e **gestione di container**. La scelta di utilizzare tale piattaforma è dovuta alla sua capacità di essere **self-healing** (*auto-placement*, *auto-restart*, *auto-replication*, *auto-scaling* dei container).

Minikube

È un tool per l'**esecuzione di un cluster** di *Kubernetes* in **locale**.

Autoscaling

Horizontal Pod Autoscaler

- Permette di scalare il numero di repliche dei Pod.
- L'algoritmo utilizzato si basa sull'osservazione dell'**utilizzazione della CPU**.
- Il numero di repliche viene calcolato con:

$$desired\_replicas = \left\lceil current\_replicas \cdot \left(\frac{current\_metric\_value}{desired\_metric\_value} \right) \right\rceil$$

- Il **numero minimo** di repliche impostato per ogni microservizio è di **una**, mentre il **numero massimo** è di **cinque**.
- L'unità di misura utilizzata da *Kubernetes* per la misurazione dell'utilizzo della CPU è il **millicore** (1000m = 1 Core)
- Nel nostro caso la **soglia di utilizzazione** è posta a 200m (1/5 di 1 Core) per il Server Web mentre per gli altri microservizi è di 100m (1/10 di 1 Core).

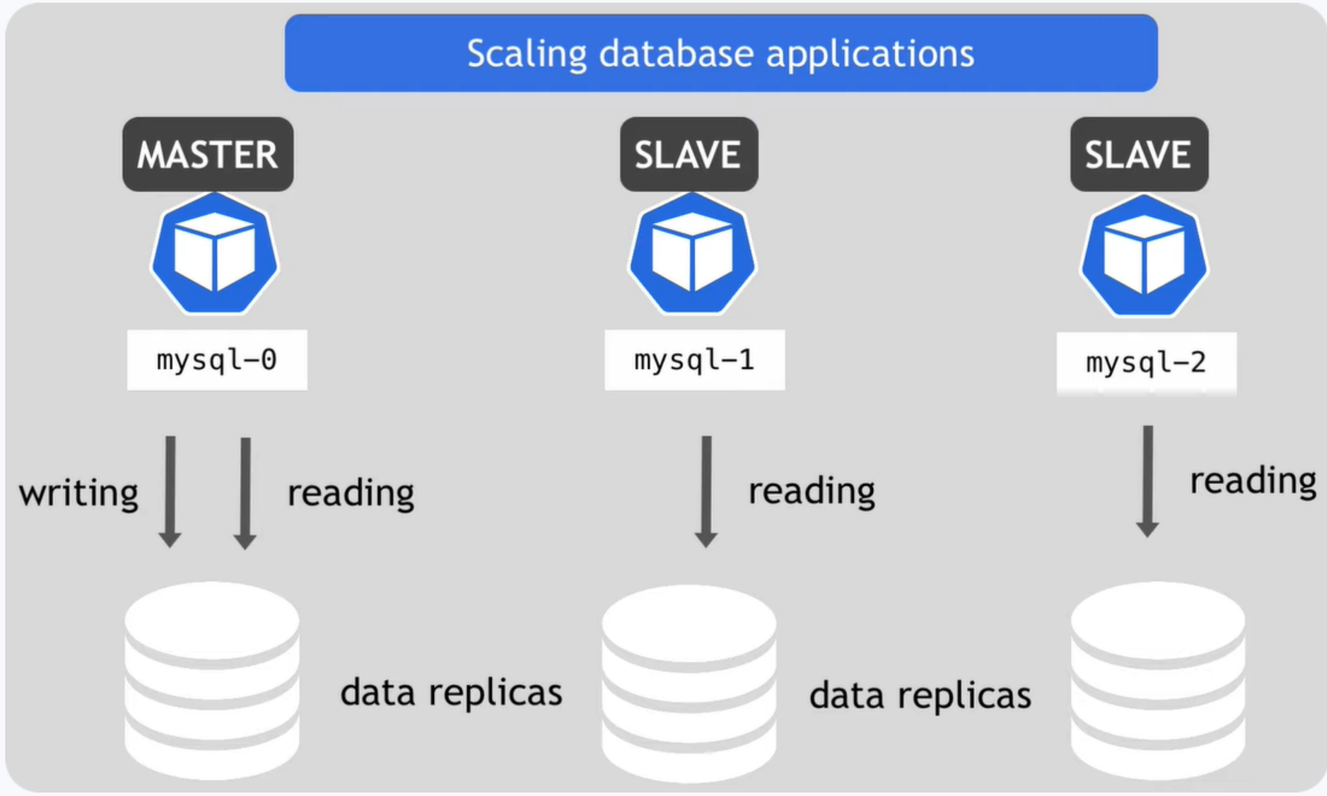
Helm Chart

È un **package manager** che consente di:

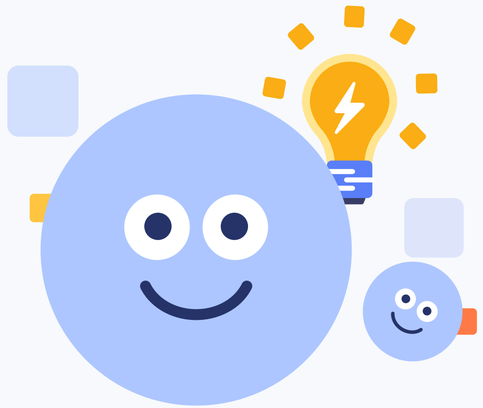
- Recuperare pacchetti software da repository
- Configurare il deploy del software
- Installare e aggiornare software e le relative dipendenza

Nell'applicazione è stato utilizzato per l'installazione del package ***MySQL*** di ***Bitnami***.

Tale package fornisce dei file yaml configurabili per la **replicazione del database** relazionale MySQL. La replicazione in questione viene effettuata creando **una replica primaria** su cui vengono effettuate tutte le scritture e **diverse repliche secondarie** per le letture.



Design Pattern



Per lo sviluppo dell'architettura a microservizi, sono stati rispettati **due design pattern**:

Circuit Breaker

Per prevenire che un malfunzionamento di un determinato microservizio si ripercuota in cascata sugli altri che vanno ad invocarlo, è stato adottato il patter *Circuit Breaker*.

- **Massimo numero di fallimenti consecutivi:** 2
- **Timeout:** 5 secondi

Database per Service

Per gestire lo stato legato ai diversi microservizi, è stato adottato il pattern *Database per Service*. Tale pattern prevede l'utilizzo di un database privato per ogni microservizio che lo necessiti.

- **Database utilizzato:** MySQL
- **Tipologia:** Database-server-per-service

Librerie

Flask

- ***flask***
- È un framework *Python* per lo sviluppo di applicazioni web

gRPC

- ***grpcio***
- È uno strumento *Python* che include il compilatore di *protocol buffer* ed un plugin per la generazione dei codici server e client a partire dai file *.proto*

RabbitMQ

- ***pika***
- È un'implementazione in *Python* per il protocollo *AMQP* utilizzato da *RabbitMQ*

RestAPI

- ***requests***
- Permette l'invio di richieste *HTTP* e di interagire con *API REST*

Circuit Breaker

- ***pybreaker***
- È un'implementazione in *Python* del design patter *Circuit Breaker*

SDCC

Grazie per l’attenzione

Enrico D’Alessandro
Alessandro De Angelis
Pierpaolo Spaziani